计算机科学1997Vol.24 №.3

一种新的平台独立的通信开发环境 xkernel

何 丹 黄 皓 金志权 谢 立
(南京大学计算机科学与技术系 南京210093)

摘 要 This paper describes a new platform-independent developing environment for constructing and composing network protocols called the x-kernel. We describe an overview of the architecture and principle of the implementation. We give the experience of developing our network system.

关键词 Communication, Networks, Developing environment

1 前 言

网络软件管理着不同进程之间交换信息的通信硬件,它必须隐藏硬件信息,处理通信中的差错,确保消息从一台机器上传送到另一台机器上的应用进程;还必须对数据编码与解码.为了有效地管理这种复杂的工作,网络软件分成多个层次,通常称为协议.协议的各层反映通信的一个方面.一般网络软件在操作系统的内核中实现.

很多操作系统支持网络协议的抽象,如 BSD UNIX的 SOCKET,SYSTEM V 的流机制.这种抽象是有用的,因为它提供了对不同协议的公共接口,从而简化了协议的构造任务.但是要定义一种一般的协议抽象是困难的,因为协议有不同种类型,如面向连接的与无连接的,同步与异步,可靠与不可靠,面向流的和面向消息的等.

为了定义网络协议的抽象对象,必须有一个子系统支持这些对象间的交互.具体地说必须把这些抽象的对象映射成进程和过程,把这些对象按功能层次结构来组织是一种好的设计技术.因为协议是分层的,所以相应的通信对象也应是分层的.但这种层与层之间的层次结构带来的开销对系统的性能有很大的影响[1].

这种把协议用对应的进程或过程调用模块来实现的方法影响了协议实现的性能.此外,还有其他一些因素影响协议的性能.如每个协议包的大小,流量控制算法,缓冲区管理及报头分析的开销等.特别是好的缓冲区管理方案,能减少协议之间数据的不必

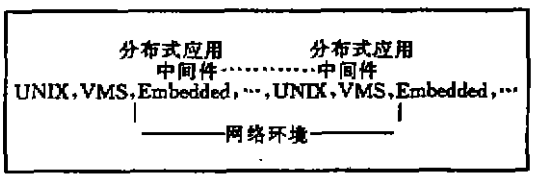
方面的相似性,不少人认为,它们的联合影响有可能克服目前在软件开发方法方面存在的缺乏学科纪律的现象.

⊙软件体系结构在软件开发和软件重用中的作用 不存在对所有领域都适用的软件体系结构.重视领域专用的软件体系结构的定义和实现,并注意同 Ada 的结合.Lockheed Martin 战术防御系统的 ADGER(Domain Architecture-based Generation for Ada Reuse)就是一个代表.提供软件重用,体系结构驱动(系统化)战略会比所谓"机会主义"战略(不做体系结构假设)得到更多的回报.

⊙面向对象的研究 如何用 Ada95和面向对象方法开发的应用系统的经验和技术.

⊙中间件技术 中间件是软件的一个层次,位于操作系统和应用之间,隐藏了操作系统、硬件平台和网络协议的复杂性,以使开发者专注于他的应用.

TRW Universal Network Architecture Services (TRW UNAS)for Ada 就是一个用于快速开发面向对象分布式系统的中间件产品.在此开发环境中,开发者可进行 Ada 和 C++的无缝编程.它通过简单的应用程序界面(API)提供分布式计算服务.



Rational 公司也有一个类似的产品.叫 UNAS Ada,是分布式系统的快速应用开发的面向对象中间件.支持多平台、协议和标准的"plug and play(即插即用)".

要拷贝。

本文将介绍和分析平台独立的协议开发环境xkernel的体系结构[1]和实现原理。

xkernel是ARIZONA大学开发的。它有三个主要的特征：1)定义了一组统一的协议抽象接口。2)使得协议对象之间的交互更加有效。3)提供了构造协议的一组公共子程序。xkernel在不同的系统平台上得到了应用。如MACH[4],SUN OS等。

2 xkernel的体系结构

传统的通信协议的实现有多种方法，如BSD的过程调用用来实现协议的各层而AT&T的SVR UNIX是用流机制(stream)来实现协议。xkernel用面向对象的方法来抽象通信协议。在这个模型中xkernel提供三类通信对象：协议对象(protocol)，会话对象(session)和消息对象(message)。这些通信对象可以是动态的或静态的，主动的或被动的。每个协议对象对应于传统网络协议，如IP,TCP,UDP等。协议之间的关系在编译时刻可配置成协议图，如图1(a)。会话对象是被动的，动态产生的。一个会话对象是协议对象的一个实例。会话对象包括对协议的解释，在它的数据结构中保存了当时网络连接时的状态信息。消息对象是主动型对象，它通过会话对象和协议对象传递协议头和用户数据，如图1(b)表示协议图1(a)的xkernel对象表示。其中□表示协议对象，○表示协议对象的实例即会话对象，TCP-IP-ETH表示虚电路连接，消息用线程表示，沿协议对象和会话对象访问。这些对象的实现将在第三节中给出。

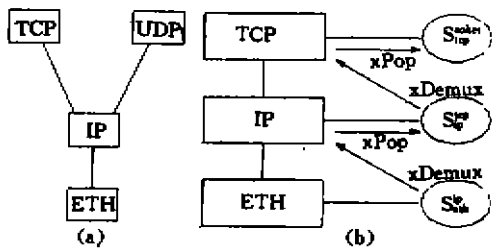


图1

2.1 协议对象

协议对象(protocol object)有两个主要功能：产生会话对象和过滤来自网络的消息并传递成一个协议对象。协议对象支持三种产生会话对象的操作：

session=xopen(hlp,hlptype,llp,participant_set)

xopen_enable(hlp,llp,participant_set)

session=xopen_done(hlp,llp,participant_set)

高层协议调用底层协议对象的xopen操作产生session。这个session是底层协议类的一个实例，hlptype代表高层协议类型，一般而言高层协议对象hlp与高层协议类型hlptype是同一个类型。在系统构成时每个协议对象都被赋予对底层协议对象操作的能力。高层协议对象对自己的能力可以通过xopen_enable传递给底层协议对象，为以后的xopen_done操作所用。当底层协议对象已为高层协议对象产生一个session时，调用高层协议对象的xopen_done操作，告诉上层协议对象session已经产生。第一个session是由用户进程来产生的，其后session的产生是由连接过程中消息传递触发产生的。participant_set是一次通信连接中每个参与者的地址集合，这个集合代表host地址、端口号、协议号等外部标识或该协议对象的能力。内部标识与外部标识之间的关系由Map子程序来绑定(binding)并以表格的形式存放在xkernel的私有数据区内。

举例说明session的建立过程。高层协议对象TCP，底层协议对象IP，参与者的集合为p={本地port，远地port}。TCP协议对象调用IP的xopen产生会话对象叫S_{tcp}^{ip}会话对象。xkernel把$p\rightarrow s_{tcp}^{ip}$的映射放入一张表中。在系统自举时TCP协议对象应调用IP协议对象的xopen_enable操作把自己的能力传递给IP协议对象，其中参与者p'={本地port}。open_enable把{本地port}→TCP的映射存入表中。产生的关系如图2。除产生session外，每个协议能把来自网络的消息通过一个过滤程序xdemux送给其中的一个会话过滤程序xdemux(protocol,message)以消息作为参数，或把消息送给已产生的session或用来触发协议对象产生新的session，并用xopen_done操作告知上层协议session已经建立好。举例说明xdemux操作，若有下层协议触发IP的xdemux操作。首先从消息头中取出本地port和远地port，并构成关系(本地port，远地port)。查找xkernel映射表知道有一个会话对象S_{tcp}^{ip}存在，因此把消息送给S_{tcp}^{ip}处理。若在表中未查到S_{tcp}^{ip}，则试查本地port在表中的映射为TCP，因此IP的xdemux调用TCP的xopen_done操作，产生一个S_{tcp}^{ip}的会话对象，并把(本地port，远地port)$\rightarrow S_{tcp}^{ip}$的映射关系存入表中，同时把消息传给S_{tcp}^{ip}处理。如图2表示协议对象与会话之间的关系。

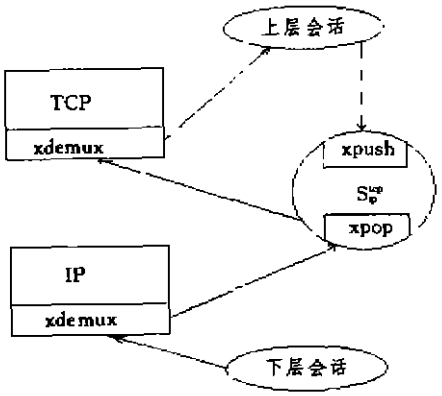


图2

2.2 会话对象

会话对象是协议对象的一个实例。在运行时刻动态地产生，在会话对象中，包括了对协议的具体解释和连接时刻的状态信息，如 TCP 会话对象，实现滑动窗口算法和相关的消息缓冲区，IP 会话对象实现数据包的拆装。

会话对象支持两种对消息的操作：xpush(session，message)和 xpop(session，message)。xpush 将高层会话对象的消息向下层会话对象传递，而 xpop 被 xdemux 调用，将下层消息往上层会话对象传送，从而实现消息从用户程序到网络驱动程序或从网络驱动程序到用户程序的消息流动。

2.3 消息对象

在 xkernel 中，消息对象是个主动对象。当消息自上而下从用户进程到网卡驱动，访问会话对象时，分两种情况处理。在同步协议中，大消息被拆装成多片，并加上协议规定的消息头。在异步协议中，某个消息可以被挂起等待另一个消息的应答。当消息自下而上传递时，消息可能为等待组装成大消息而挂起或与等待伙伴消息串行执行。

以上是 xkernel 的构造模型及体系结构，下面我们重点讨论 xkernel 在实现上的原理。

3 xkernel 的实现原理

xkernel 的一个重要特征是定义了一个明确的协议结构。这个结构主要从两个方面考虑：一是把网络协议划分成两个正交的组成部分：协议对象与会话对象。并提供构造协议所需的统一的协议接口。其中协议对象用于切换相应会话对象所需的消息。会话对象用于协议的解释。这样区分是为了便于协议代码的书写和理解。二是 xkernel 提供缓冲区管理、映射管理、事件管理和线程管理来支持协议的实现。在 unix 系统的网络实现中，这种对协议支持的实现，常常是凭经验的。因此 xkernel 有三大优点：一是不必专门优化就能有效地实现协议。二是由于结构明确，xkernel 可以提供一套统一的独立于协议的接口。三是提供了一种动态配置网络协议的实现环境，具有较好的模块性和可配置性。在编译时刻，用一种元语言书写协议图，可以支持不同的协议拓扑。

3.1 统一的协议接口

xkernel 的每个协议用统一的协议接口(UPI)表示。在运行时表现为协议对象和会话对象。为实现协议对象和会话对象，xkernel 引入了一个重要的对象 xobj 数据结构。在这个数据结构中，对象的方法分别是实现协议对象和会话对象的操作。私有变量代表静态协议名和动态连接时的状态。为了编程上的方便。xkernel 把协议分成五类：

一、异步协议，如 IP，TCP 和 UDP。可用 xpush，xpop 及 xdemux 操作传达消息。异步协议是对称的，即每个会话对象都能处理输入/输出消息。这种对称性增强了构造协议的能力，使实现特定的协议变得容易。

二、同步协议，如 RPC 协议。这种协议是不对称的，客户方与服务方的会话是不同的。xkernel 用 xcall，xcallpop 和 xcalldemux 支持消息的传送，其中 xcall 用于客户方向服务方请求消息，xcallpop，xcalldemux 用于服务方回答消息。

三、控制协议，如 ARP，ICMP。对这种协议的消息传输，既不用同步协议操作，也不用异步协议操作，只能用 xcontrol 发送控制消息。

四、接口协议。由于 xkernel 独立于运行平台，协议的实现最终要与系统的网络驱动和用户程序有接口。接口协议的作用在于实现协议体上下端与网络硬件如周边系统的无缝连接。

五、虚协议。可以在协议图的任何位置，由虚协议对象产生的会话，不会对网络消息加头或解释处理。它起路由作用，决定协议通过会话的路径和消息的大小等特性。虚协议支持同步或异步的操作。

3.2 协议的支持环境

为了实现 xkernel 的协议环境，有一组公共的子程序库支持协议的实现。其中缓冲管理子程序用堆数据结构来实现，维护消息的缓冲区分配、截断、合并等操作。

映射库程序，用于标识符间的映射维护。它支持

添加一个新的映射、删除、变换、查找等操作，在实现协议时可用这些操作把消息头的地址，端口域翻译成相应的标识符。协议一般维护主动映射和被动映射。主动映射包含当前主动连接的信息，用于处理输入会话的操作。如 open 操作可以建立协议与 active 键的映射。被动映射用于标识符与 Enable 对象的关系。open_enable 操作可以建立 passive 键与 Enable 对象的关系。

事件库提供定时器设施，它允许一个协议在特定时刻自启动一个程序执行，特别适用于超时处理或等待消息。

xkernel 使用每个消息一个线程的计算模型。线程库提供了一组线程调度的同步原理。线程库的读/写锁同步机制适合于多处理机的计算环境，xkernel 的线程是非抢占的，因为每个协议是独立的成分。当它被高层协议调用时，它就放弃占有处理机，协议之间的操作都可引起线程的切换。线程也不能被阻塞。如 RPC 协议，或在 IP 协议中当一个消息是一个报文的一片时，就阻塞线程等待下一片数据的到来，组装成一个报文。

3.3 xkernel 的初始化

在编译 xkernel 之前，首先定义一个协议图和协议号。协议图由 xkernel 的元语言描述。如图1(a)的协议图定义如下：

```
name=ethdrv/eto
name=eth        protocols=ethdrv
name=IP         protocols=eth
name=TCP        protocols=IP
name=UDP        protocols=IP
```

用 xkernel 的 compose 程序生成协议图初始化程序，在启动 xkernel 时，由这个程序产生协议对象的第一个会话对象。这种灵活的可配置性是任何其他协议构造方法都不能比拟的。

结束语 xkernel 与 BSD 过程调用和 SVR4.0 的流机制为背景的网络服务器相比，发现 BSD 在网络实现上是每个协议一个进程，不支持同步发送，当一个发送者进程异步发出消息就阻塞自己，处理机切换到另一层协议进程处理消息，xkernel 是每个消息一个线程。处理消息时在同一个地址之间执行不必进行处理机切换，特别适合于多处理机环境。SVR4.0在协议实现上基于流机制，流机制必须有操作系统内核支持。Mach3.0是微内核的操作系统环境，不宜在内核实现流机制。因此，在 Mach3.0核外实现流机制，在此基础上实现网络服务器开销太大，而且流机制对除网络协议之外的系统开发并不带来多大的好处。xkernel 是一种平台独立的通信开发环境，既可以运行在内核地址空间，又可运行在用户地址空间，给开发者带来极大方便。

在本文写作的过程中，还得到杨培根教授的帮助和研究生钱晓燕、蒋臻同学的讨论，他们提出不少建设性的建议，在此表示感谢！

参考文献

[1] N. C. Hutchinson, L. L. Peterson, The xkernel: An architecture for implementing network protocols, IEEE Trans. on SE., 17(1), 1991

[2] L. L. Peterson, B. S. Davie, A. C. Bavier, xkernel Tutorial, Jan. 1996

[3] A. Habermann et al., Modularization and hierarchy in a family of operating system, Commun. ACM, 19(5), 1976

[4] Frano Travostino, Real-time local and remote MACH ipc: Architecture and design, Version 1.0(DRAFT), OSF Research Institute Advance Development, 1994.4

[5] Edwin F. Menze Ⅱ, H. K. Ormen, xkernel programmer's manual, 1996